## Remarks

This REPLY is in response to the Office Action mailed June 5, 2009.

### I. Summary of Examiner's Rejections

Prior to the Office Action mailed June 5, 2009, Claims 1-4, 6, 7, 9, 11-14, 16, 17, 19, 31-34 and 43-46 were pending in the Application. In the Office Action, Claims 1-4, 6, 7, 9, 11-14, 16, 17, 19, 31-34 and 43-46 were rejected under 35 U.S.C. 103(a) as being unpatentable over Taylor et al. (U.S. Publication No. 2004/0019897, hereafter Taylor) in view of Susarla et al. (U.S. Patent No. 6,915,511, hereafter Susarla) and further in view of Peterson (Brett Peterson, Understanding J2EE Application Server Class Loading Architectures).

### II. Summary of Applicant's Amendment

The present Reply amends Claims 1, 11 and 43-46; leaving for the Examiner's present consideration Claims 1-4, 6, 7, 9, 11-14, 16, 17, 19, 31-34 and 43-46.

### III. Claim Rejections under 35 U.S.C. 103(a)

In the Office Action mailed June 5, 2009, Claims 1-4, 6, 7, 9, 11-14, 16, 17, 19, 31-34 and 43-46 were rejected under 35 U.S.C. 103(a) as being unpatentable over Taylor (U.S. Publication No. 2004/0019897) in view of Susarla (U.S. Patent No. 6,915,511) and further in view of Peterson (Brett Peterson, Understanding J2EE Application Server Class Loading Architectures).

**Claim 1**

Applicant respectfully traverses the rejection of Claim 1 for at least the reasons provided below. To expedite prosecution, Claim 1 has also been amended to recite:

> 1. *(Currently Amended) A system for loading software applications, comprising:*
>
> *a server, executing a Java virtual machine that includes a system classloader, for storing and running a plurality of software applications, wherein each of said plurality of software applications includes a plurality of deployable modules and classes associated therewith, and wherein the software applications or modules therein can be customized by a*

*software developer and then deployed to run on the same server;*

*a control file, that can be edited by the software developer and associated with said plurality of software applications or modules therein, wherein said control file specifies a hierarchy of application classloaders as children of the system classloader, to be used with the modules in said plurality of software applications, and wherein the hierarchy includes a plurality of nested branches, including providing each of said plurality of software applications or modules therein with its own application classloader hierarchy so that the software applications and modules are not aware of classloaders or classes that are assigned to another software application, and wherein the hierarchy is specified by the software developer to provide namespace separation between two or more of the plurality of software applications or between different modules in any one of the software applications; and*

*a deployment utility that, upon receiving a request to deploy and run a software application on the server,*

*parses the control file and determines which classloaders are specified therein for the software application being deployed,*

*loads with said software application into the Java virtual machine at the server a selection of said application classloaders corresponding to the hierarchy specified by said control file, including, if a particular software application or a module in a software application is being redeployed then loading only the application classloaders that are specified in the branches for that particular software application or module, without loading any of the other branches in the hierarchy, and*

*enables the server to host multiple isolated software applications or modules within the Java virtual machine, as defined by the hierarchy.*

Taylor discloses a method, system, and program for processing objects in a distributed computing environment. (Abstract). As disclosed therein, the population manifest 326 is parsed by the container 306 to construct class loaders as necessary to load the components indicated in the manifests. FIG. 6 illustrates an example of a class loader hierarchy 350 that the population manifest 326 may implicitly define. The class loader hierarchy 350 divides class loaders into factory, root, and component class loaders. Component class loaders are used to load the variable components of the system, such as services, facility implementations, plug-ins, and dynamically

loaded modules. Root class loaders load common interfaces that enable communication between the components of the system, such as facility interfaces. Factory class loaders are used to load factory classes. A factory is a component responsible for instantiating and destroying a particular type (or set of types) of components. A facility is a component whose instances can be shared across other components. (Paragraph [0050]). Each element in the one or more XML files forming the population manifest 326 indicates a component, encapsulated in a CAR file, to be loaded, and the attributes of the element may include information about the component, such as public package, version, etc. During runtime, a new component may be added to the system by adding a new element to the XML file comprising a population manifest 326. A root XML file may maintain information on the root components that are loaded, where the last XML element in the root XML file would comprise the component that is loaded by the lowest root class loader in the hierarchy. (Paragraph [0052]).

Susarla discloses providing a dynamic class reloading using a modular, pluggable and maintainable class loader. In one embodiment, to enforce module level separation of the utility classes, the class loader stack may include another layer between the application class loader 202 and the other layers. This layer may load all the classes that are visible only to a module but not cross-module. The table of FIG. 3 illustrates the interdependency between the classes loaded by the class loaders in the class loader stack illustrated in FIG. 4, according to one embodiment. (Column 11, Lines 7-14).

Peterson describes an overview of different proprietary class loading architectures, as provided by different application server vendors. For example, Peterson describes that, in WebSphere 4.0, different isolation modes can be used, including Module: One class loader is created for each module in an .ear. A module is defined as a web app .war, an EJB .jar, or a .jar referenced from the Manifest Class-Path of a .war or .jar. The logical class loader hierarchy is formed by dependencies specified in the Manifest Class-Path attributes of the modules. For an application with two .war files, two EJB .jar files, and two common .jar utility libraries listed in Manifest Class-Path attributes, six class loaders would be created. (Figure 3 and related text).

Peterson also describes, for example, that in HP-AS 8 Maintenance Pack 3, the library class loader is the parent of all application class loaders. The classpath it searches is specified by entries in the proprietary application-classloader-service-config.xml file typically found in <HP-AS-

HOME>/config/hpas. (Figure 4 and related text). One application class loader is created as a child of the library class loader for each J2EE application deployed in HP-AS 8. One EJB class loader is created as a child of the application class loader. The EJB class loader is responsible for loading all classes within all EJB .jar archives in the J2EE application. One web application class loader is created as a child of the application class loader for each web application archive in the J2EE application. The fact that each .war gets its own class loader allows web application independence to be achieved within an application. (Figure 4 and related text).

In the Office Action mailed June 5, 2009, it was asserted that Taylor in view of Susarla discloses substantially the features recited by Claim 1. It was apparently acknowledged that neither Taylor nor Susarla explicitly disclose that the hierarchy includes providing each of said plurality of software applications with its own application classloader hierarchy so that the software applications are not aware of classloaders or classes that are assigned to another software application; however, it was asserted that Peterson discloses a hierarchy class loading architecture with isolation mode; and further discloses a class loading architecture whereon a web application class loader is created as a child of the application class loader; and that it would have been obvious to one having ordinary skill in the art at the time the invention was made to use Peterson's class loading hierarchy architecture to load the classes of Taylor to a JVM in local server to enable application independence.

Applicant respectfully submits that, based on the above descriptions, while Taylor, Susarla, and Peterson appear to disclose the use of class loaders and/or various means of configuring different systems to support different class loader modes, none of the cited references appear to disclose or render obvious several features recited by Claim 1, for example: a control file, that can be edited by the software developer and associated with said plurality of software applications, wherein the control file specifies a hierarchy of application classloaders as children of the system classloader, to be used with the modules in said plurality of software applications, and wherein the hierarchy includes a plurality of nested branches, including providing each of said plurality of software applications with its own application classloader hierarchy; or a deployment utility that loads with said software application into the Java virtual machine at the server a selection of said application classloaders corresponding to the hierarchy specified by the control file.

A benefit of such features recited by Claim 1 is that the software developer is provided with

more control over the granularity of module redeployment, and can explicitly declare how module level classloaders in an application are organized. This granularity assists in the task of software development, including reducing the need for the software developer to redeploy all of the modules of an application each time a modification is made to a single module.

Applicant respectfully submits that, although Peterson appears to describe a number of different class loading architectures, both Peterson and the other cited references appear to describe class loader architectures that, for a particular architecture, use a particular method of class loading.

For example, Peterson appears to describe that, in WebSphere 4.0's module isolation mode, one class loader is created for each module in an .ear. This appears to indicate that, using that particular architecture a separate class loader is created for each module. Similarly, Peterson appears to describe that, in HP-AS 8, one EJB class loader is responsible for loading all classes within all EJB .jar archives in the J2EE application, while one web application class loader is created as a child of the application class loader for each web application archive in the J2EE application. This appears to indicate that, using that particular architecture the class loaders are always configured in that manner.

However, Applicant respectfully submits that neither Taylor, Susarla, nor Peterson, when considered alone or in combination, appear to disclose the use of a control file or configuration file, that can be edited by the software developer and associated with said plurality of software applications or modules, wherein said control file specifies a hierarchy of application classloaders as children of the system classloader, to be used with the modules in said plurality of software applications, and wherein the hierarchy includes a plurality of nested branches, including providing each of said plurality of software applications or modules with its own application classloader hierarchy so that the software applications or modules are not aware of classloaders or classes that are assigned to another software application, and wherein the hierarchy is specified by the software developer to provide namespace separation between two or more of the plurality of software applications or between different modules in any one of the software applications. Claim 1 has been amended to more clearly recite these features.

In view of the above comments, Applicant respectfully submits that Claim 1, as amended, is neither anticipated by, nor obvious in view of the cited references, when considered alone or in

combination. Reconsideration thereof is respectfully requested.

**Claims 11 and 45**

The comments provided above with regard to Claim 1 are herein incorporated by reference. Claims 11 and 45 have been amended similarly to Claim 1. For similar reasons as provided above with respect to Claim 1, Applicant respectfully submits that Claims 11 and 45, as amended, are likewise neither anticipated by, nor obvious in view of the cited references, when considered alone or in combination. Reconsideration thereof is respectfully requested.

**Claims 43-44 and 46**

Claims 43-44 and 46 depend from and include all of the features of Claim 1, 11 or 45. Applicant respectfully submits that these claims are allowable at least as depending from an allowable independent claim, and further in view of the amendments to the independent claims, and the comments provided above.

Additionally, in the Office Action mailed June 5, 2009, it was asserted that Taylor discloses wherein the control file includes a tag layout, and application class-loader structure elements, that determine the hierarchy of modules and classes of the software application to be loaded into the application server; and wherein upon deployment, the deployment utility retrieves modules and classes from a computer readable medium in a manner consistent with the tag layout in the configuration file, and constructs an application container at the server with the classes and modules, in the order in which the classes and modules were retrieved, to create a hierarchical classloader.

However, Applicant respectfully submits that, while Taylor appears to disclose using an XML file to determine factory components to be hosted in a container, so that each specified instance is instantiated in the order the instance appears in the file, Taylor does not appear to disclose the use of a tag layout to declare the application class-loader classloader hierarchy, an example of which is illustrated in Applicant's Specification at Paragraphs [0073-0074].

Claims 43-44 and 46 have been amended to more clearly recite these features. Reconsideration thereof is respectfully requested.

**Claims 2-4, 6, 7, 9, 12-14, 16, 17, 19 and 31-34**

Claims 2-4, 6, 7, 9, 12-14, 16, 17, 19 and 31-34 depend from and include all of the features of Claim 1, 11 or 45. Applicant respectfully submits that these claims are allowable at least as depending from an allowable independent claim, and further in view of the amendments to the independent claims, and the comments provided above. Reconsideration thereof is respectfully requested.

## V. Conclusion

In view of the above amendments and remarks, it is respectfully submitted that all of the claims now pending in the subject patent application should be allowable, and reconsideration thereof is respectfully requested. The Examiner is respectfully requested to telephone the undersigned if he can assist in any way in expediting issuance of a patent.

Applicant believes no fee is due with the communication. However, the Commissioner is authorized to charge any underpayment or credit any overpayment to Deposit Account No. 06-1325 for any matter in connection with this response, including any fee for extension of time, which may be required.

Respectfully submitted,

Date: August 5, 2009

By: /Karl Kenna/
Karl Kenna
Reg. No. 45,445

Customer No.: 23910
FLIESLER MEYER LLP
650 California Street, 14th Floor
San Francisco, California 94108
Telephone: (415) 362-3800